US-PAT-NO: 6615319

DOCUMENT-IDENTIFIER: US 6615319 B2

TITLE: Distributed mechanism for resolving cache **coherence** conflicts in a multi-node computer architecture

DATE-ISSUED: September 2, 2003

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Khare; Manoj	Saratoga	CA	N/A	N/A
Looi; Lily P.	Portland	OR	N/A	N/A
Kumar; Akhilesh	Sunnyvale	CA	N/A	N/A
Briggs; Faye A.	Portland	OR	N/A	N/A

US-CL-CURRENT: 711/141, 711/124 , 711/146

ABSTRACT:

According to one embodiment, a method is disclosed. The method comprises receiving a read request from a first node in a multi-node computer system to read data from a memory at a second node. Subsequently, a write request from a third node is received to write data to the memory at the second node. The read request and write request is detected at conflict detection circuitry. Finally, read data from the memory at the second node is transmitted to the first node.

21 Claims, 7 Drawing figures

Exemplary Claim Number: 1

Number of Drawing Sheets: 7

---------- KWIC ---------

TITLE - TI (1):
Distributed mechanism for resolving cache **coherence** conflicts in a multi-node computer architecture

Brief Summary Text - BSTX (2):
The present invention relates to computer systems; more particularly, the present invention relates to resolving cache **coherence** conflicts in a computer system.

Brief Summary Text - BSTX (4):
In the area of distributed computing when multiple processing nodes access each other's memory, the necessity for memory **coherency** is evident. Various methods have evolved to address the difficulties associated with shared memory environments. One such method involves a distributed architecture in which each node on the distributed architecture incorporates a resident **coherence** manager. Because of the complexity involved in providing support for various protocol implementations of corresponding architectures, existing shared memory multiprocessing architectures fail to support the full range of **MESI** protocol possibilities. Instead, existing shared memory multiprocessor architectures rely on assumptions so as to provide a workable although incomplete system to address these various architectures.

Brief Summary Text - BSTX (5):
One of the fundamental flaws of these existing memory sharing architectures is that a responding node, containing modified data for a cache line where the home storage location for the memory in question resides on a different node, is expected only to provide a passive response to a read request. No mechanism is built into the architectures to provide intelligent handling of the potential conflict between back-to-back read and write requests to the same line of memory. Therefore, a distributed mechanism for resolving cache **coherence** conflicts in a multiple processing node architecture is desired.

Drawing Description Text - DRTX (5):
FIG. 3 is a flow diagram for one embodiment of cache **coherence** for a memory read command at a computer system;

Detailed Description Text - DETX (2):

A method and apparatus for resolving cache **coherence** conflicts in a multi-node computer architecture is described. In the following detailed description of the present invention numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the present invention.

Detailed Description Text - DETX (5):

A scalability port (SP) is an inter-node interface used to enable the implementation of a shared memory architecture, multi-processor system. The scalability port is a point to point cache **coherent** interface for interconnection of processor nodes 105 with local memory, I/O nodes 120 and network switches. Cache **coherence** is a mechanism to provide a consistent view of memory in a shared memory system with multiple caching agents that could have copies of data in private caches. Any updates to the memory block must be done in a manner that is visible to all of the caching agents. Although computer system 100 has been shown with three processor nodes and one I/O node, computer system 100 can be implemented with other quantities of processor and I/O nodes.

Detailed Description Text - DETX (7):

The link layer abstracts the physical layer from the protocol layer, thus, guaranteeing reliable data transfer between agents on a SP. In addition, the link layer is responsible for flow control between the two agents on a SP and provides virtual channel services to the protocol layer. Virtual channels allow sharing of the physical channel by different protocol level messages for cache **coherence**.

Detailed Description Text - DETX (8):

The protocol layer implements the platform dependent protocol engines for higher level communication protocol between nodes such as cache **coherence**. According to one embodiment, the protocol layer uses packet based protocol for communication. The protocol layer formats a packet (e.g., request, response, etc.) that needs to be communicated and passes it to the appropriate virtual channel in the link layer. The protocol layer is bypassed in pure routing agents resulting in low latency transfer from sender to the receiver through the network.

Detailed Description Text - DETX (14):

SP switch 230 operates according to a central **snoop coherence** protocol. The central **snoop coherence** protocol is an invalidation protocol where any caching agent that intends to modify a cache line acquires an exclusive copy in its cache by invalidating copies at all the other caching agents. The **coherence** protocol assumes that the caching agents support some variant of a **MESI coherence** protocol, where the possible states for a cache line are Modified, Exclusive, Shared or Invalid.

Detailed Description Text - DETX (15):

The **coherence** protocol provides flexibility in **snoop** responses such that the protocol layer at the SP switch 230 can support different types of state transitions. For example, a cache line in the Modified state can transition either to a Shared state on a remote **snoop** or an Invalid state on a remote **snoop, and the snoop** response on the SP can indicate this for appropriate state transitions at SP switch 230 and the requesting agent. SP switch 230 includes a **snoop** filter (not shown). The **snoop** filter is organized as a tag cache that keeps information about the state of a cache line and a bit vector (presence vector) indicating the presence of the cache line at the caching nodes. In one embodiment, the presence vector has one bit per caching node in the system. If a caching agent at any node has a copy of the cache line, the corresponding bit in the presence vector for that cache line is set. A cache line could

be
either in Invalid, Shared, or Exclusive state in the **snoop** filter.

Detailed Description Text - DETX (16):
According to a further embodiment, the **snoop** filter is inclusive (e.g., without data, only the tag and state) of caches at all the caching agents. Thus, a caching agent does not have a copy of a cache line that is not present in the **snoop** filter. If a line is evicted from the **snoop** filter, it is evicted from the caching agents of all the nodes (marked in the presence vector). In other embodiments where multiple SP switches 230 may be included, the **snoop** filter is divided amongst the multiple SP switches 230 or into multiple caches within one switch 230 in order to provide sufficient **snoop** filter throughput and capacity to meet the system scalability requirement. In such embodiments, different **snoop** filters keep track of mutually exclusive set of cache lines. A cache line is tracked at all times by only one **snoop** filter.

Detailed Description Text - DETX (17):
The state of a cache line in the **snoop** filter is not always the same as the state in the caching agents. Because of the distributed nature of the system, the state transitions at the caching agents and at the **snoop** filter are not synchronized. Also, some of the state transitions at the caching agents are not externally visible and therefore the **snoop** filter may not be updated with such transitions. For example, transitions from Exclusive to Modified state and replacement of cache lines in Shared or Exclusive state may not be visible external to the caching agent.

Detailed Description Text - DETX (18):
In the Invalid state, the **snoop** filter is unambiguous. Thus, the cache line is not valid in any caching agent. All bits in the presence vector for the line in the **snoop** filter are reset. An unset bit in the presence vector in the **snoop** filter for a cache line is unambiguous. Consequently, the caching agent

at the node indicated by the bit does not have a valid copy of the cache line.
A cache line in Shared state at the **snoop** filter may be either in Shared or
Invalid state at the caching agents at the node indicated by the presence
vector in the **Snoop** Filter. A cache line in Exclusive state at the **Snoop**
Filter may be in any (Modified, Exclusive, Shared or Invalid) state at the
caching agents at the node indicated by the presence vector in the **Snoop**
Filter.

Detailed Description Text - DETX (19):
FIG. 3 is a flow diagram for one embodiment of cache **coherence** for a memory
read **request from a node** requesting access (e.g., processor node 105a) to a
memory 215 at a node containing the requested logical address (e.g., the memory
215 at processor node 105c(or **home node**)) wherein a cache line corresponding to
the logical address of the memory 215 has been modified at a remote modified
node (e.g., processor node 105b).

Detailed Description Text - DETX (20):
Upon a read request by the **request node,** a cache line in the remote modified
node corresponding to the requested **home node** memory 215 line may have been
modified. Therefore, the cache line in the modified node is checked before the
request node reads data from the **home node**. Referring to FIG. 3, a port read
request is received at SP switch 230 from the **request node** (e.g., node 105a) at
process block 305. The port read request is used to read a cache line. In
particular, the port read is used to both read from memory and **snoop** the cache
line in the caching agent(s) at the modified node. The port read request is
targeted to the **coherence** controller or the **home node** of a memory block. A
node that is not home of the block addressed by the transaction does not
receive a port read request.

Detailed Description Text - DETX (21):
At process block 310, SP switch 230 executes a search of its

internal **snoop** filter (e.g., a **snoop** filter lookup) to determine if the modified node (e.g., node 105b) contains a modified cache line corresponding to the requested memory address. At process block 315, a speculative read request is transmitted to the **home node** (e.g., node 105c). The speculative read request is used to read the home memory 215. In one embodiment, the speculative read request can be dropped by the responding agent without any functional issue. At process block 320, a port **snoop** request is transmitted from SP switch 230 to the remote modified node. The **snoop** request is used to **snoop** a memory block at a caching node. As a result of the **snoop** request, data may be supplied to both the source node and the home memory is updated.

Detailed Description Text - DETX (22):

At process block 325, a port **snoop** result and read data is transmitted from the modified node to the SP switch 230. The port **snoop** result is used to convey the result of **snoop** back to the node A. According to one embodiment, the port **snoop** result response indicates whether the line was found in a Modified state. If the cache line is found in a modified state, the cache holds the most recent version of data. If not, the data in the **home node** is the most recent, and the cache line is invalidated. At process block 330, it is determined whether the data in the cache line has been modified.

Detailed Description Text - DETX (23):

If it is determined that the cache line at the remote modified node has been modified, the port **snoop** result and read data is transmitted from the SP switch 230 to the **request node,** process block 335. At process block 340, the memory 215 within the **home node** is updated to reflect the up to date data from the modified remote node cache. However, if the **snoop** result indicates that the state of the cache line has not been modified, the **snoop** result received at SP switch 230 is returned as invalid. As a result, the invalid **snoop** result is transmitted from the SP switch 230 to the **request node,** process block

345. At process block 350, a read access is executed at the memory 215 within the **home node**. At process block 355, the read data is transmitted from the **home node to the request node** via SP switch 230.

Detailed Description Text - DETX (24):
A read-write conflict may occur when a cache line in a node (e.g., the remote modified node) is in the Modified state. As described above, if the **request node** makes a request for a copy of the line, the **coherence** protocol must make sure that the data supplied to node A is the most current data which may be in the Modified node. However, it is possible that while the request for a copy of the cache line is being processed (e.g., after the **snoop** filter look up), the processor with the copy of the cache line at the modified node may decide to write over the cache line. If the request from the **request node** is allowed to proceed between the interval of writing over the modified line from the modified node and memory 215 update at the **home node,** node A may get a stale copy of the line from the memory 215.

Detailed Description Text - DETX (26):
Assuming that node B has the modified copy of a line and node A makes a read request for a copy. If the **request from node** A reaches the **snoop** filter in SP switch 230 before a write from node B, the read **request from node** A will initiate a **snoop request to node** B. Thus, if no conflict detection mechanism is implemented, the read request may not see the on-going write from node B and may respond to the **snoop with a snoop** result. The **snoop** result response from node B going over the response channel may bypass the write from node B going over the request channel. Once a **snoop** result from node B for the read **request from node** A is received by SP switch 230, it will read the cache line from the memory 215 at the **home node** and supply it as data to node A. The line read from the memory 215 at the **home node** does not have the most recent data. Accordingly, an incoherent system state occurs.

Detailed Description Text - DETX (27):

According to one embodiment, computer system 100 includes a conflict detection mechanism for instances where **coherent** agents in computer system 100 generate transactions addressed to the same cache line. The mechanism orders the transactions in such a way that the **coherency** is not violated. In one embodiment, the detection and resolution of conflicts among concurrent requests from multiple nodes is done at SNC 210 and SP switch 230. As described above, concurrent accesses from multiple nodes to the same cache line creates a problem if the requests are conflicting in nature. Two requests are considered conflicting with each other if simultaneous processing of these requests will cause the system to get into an incoherent state, or result in loss of most up-to-date data.

Detailed Description Text - DETX (29):

Bus interface 510 provides the interface between SNC 210 and processor bus 202. IRB 515 is used to store SP requests initiated due to requests at remote nodes. These requests could be a memory access at the node, a **snoop** access to the caching agents at the node, or a combination of both. According to one embodiment, each IRB 515 entry includes the address, request type, **snoop** result, other state information and data. In a further embodiment, the conflict detection and resolution due to concurrent accesses to the same cache line at a node requires that some IRB 515 entries are blocked for some event at a conflicting ORB 520 entry. Thus, the number of entries in IRB 515 is larger than the number of entries in ORB 520 to prevent deadlocks.

Detailed Description Text - DETX (30):

ORB 520 includes a buffer that keeps track of outstanding **coherent** requests on the SP. In particular, the ORB 520 buffer keeps track of the address, transaction identifier, local **snoop** result, **snoop** and data response, completion response and a pointer to a pending **snoop** for that address due to a request generated at a remote node. According to one embodiment, ORB 520 has

one
outstanding transaction at any time for a particular cache line.

Detailed Description Text - DETX (31):
SP switch includes a **snoop** pending table (SPT) 540 and a **snoop** filter (SF)
550. As discussed earlier, SF 550 tracks the state of cache lines in the
caching nodes. In particular SF 550 is inclusive of tags in the processor
caches and is used to filter **snoops** from showing up at remote nodes that do not
contain a copy of a particular data block. SPT 540 tracks transactions
received at SP switch 230 from all ports until **snooping** has completed. In
particular, SPT 540 orders multiple concurrent requests from different ports to
the same cache line. In one embodiment, each SP 540 entry includes the
address, the cache line state at SF 550 and the presence vector of the cache
line.

Detailed Description Text - DETX (32):
With the implementation of the conflict detection mechanism, SP switch 230
and SNCs 210, the read-write conflict scenario shown in FIG. 4 can be detected
and resolved based on the order in which SP switch 230 processes these
requests. FIG. 6A is a timing diagram for one embodiment of detecting a
read-write conflict. In this scenario, a port read request is received at SP
switch 230 from node A at time t1, while the port write request is received
from node B at time t2. The speculative read is transmitted from SP switch 230
to node B at time 3, and the **snoop** request is transmitted to the node C at time
t4. Note that the **snoop** request is blocked from the IRB 515 within node B
because of the write request for the same line being stored in the ORB 520.
Accordingly, the **snoop** request cannot be completed until an acknowledgement is
received at node B corresponding to the write request.

Detailed Description Text - DETX (33):
However, the conflict between the read and write is detected by SPT 540,
thus, the write request is rejected. Consequently, a retry response is
received back at node B at time t5. In response to receiving the retry
response, the read **snoop** request may now be completed. At times t6 and t7 a

snoop result indicating that the cache line at node B is in the Modified state
is received at SP switch 230 and node A, respectively. Since the cache line
has been modified, the data from the cache line is transmitted along with the
snoop result. At time t8 the port write stored in SPT 540 is received at node
C. At times t9 and t10 and acknowledgement that the write has been completed is
received at SP switch 230 and node A, respectively.

Detailed Description Text - DETX (35):
After the write acknowledgement, the port read is again received at SP
switch 230 at time t7. At times t7 and t8 a speculative read and read requests
are received at node C from SP switch 230. At time t9 a **snoop** result
indicating that the cache line at node B is in the Invalid state is received at
SP switch 230 and node A, respectively. At time t10 the read is received at
node A.

Claims Text - CLTX (1):
1. A method comprising: receiving a read request from a first processor
node in a multi-node computer system via a point-to-point cache **coherent**
interface to read data from a memory device at a second processor node, the
interface comprising two or more layers having a separate set of protocol
functions; receiving a write request from a third processor node via the
interface to write data to the memory device; detecting the read request and
write request at conflict detection circuitry; and transmitting data
corresponding to the read request from the memory device to the first processor
node via the interface.

Claims Text - CLTX (2):
2. The method of claim 1 further comprising transmitting a **snoop** request to
the third node upon receiving the read request.

Claims Text - CLTX (3):
3. The method of claim 2 wherein the **snoop** request is blocked at the third
node due to the pending write request.

Claims Text - CLTX (6):
6. The method of claim 2 wherein the result of the **snoop** request is transmitted to the first node with the data corresponding to the read request.

Claims Text - CLTX (9):
9. A method comprising: receiving a write request from a first processor node in a multi-node computer system via a point-to-point cache **coherent** interface to write data to a memory device at a second processor node, the interface comprising two or more layers having a separate set of protocol functions; receiving a read request from a third processor node via the interface to read data from the memory device; detecting the read request and write request at conflict detection circuitry; and writing the data from the first processor node to the memory device.

Claims Text - CLTX (12):
12. The method of claim 11 further comprising transmitting a **snoop** result to the third node after receiving the second read request.

Claims Text - CLTX (13):
13. A computer system comprising: a first processor node; a point-to-point cache **coherent** interface, coupled to the first processor node, comprising two or more layers, Each layer having a separate set of protocol functions; a scalability port (SP) switch coupled to the interface; a second processor node coupled to the SP switch; and a third processor node coupled to the SP switch wherein the first processor node, the second processor node and he SP switch comprise a conflict detection mechanism that detects conflicting request from the first processor node and the second processor node to access a memory device within the third processor node.

Claims Text - CLTX (19):
19. The computer system of claim 13 wherein the SP switch comprises: a SP interface; a **snoop** pending table (SPT) coupled to the SP interface; and a

snoop filter coupled to the SPT and the SP interface.